Álvaro Suárez. NIA: 100390789
Adrián Hernández. NIA: 100414379

Laboratorio 1: Tipos Abstractos de Datos

Diseño e implementación

Se desea implementar una clase que represente una matriz bidimensional de números reales en doble precisión que permita realizar algunas operaciones de álgebra lineal básica.

Deben ofrecerse como mínimo las siguientes funcionalidades:

- Constructor a partir de número de filas y columnas. Todos los valores de la matriz se inician a cero. La matriz almacena el número de filas y de columnas y un búfer en los que almacenan todos los elementos de la matriz. El búfer debe ser un único array unidimensional reservado en memoria dinámica.

```
matrix a{n,n};
```

- Constructor por defecto. Se crea una matriz de cero filas y cero columnas que no consume memoria dinámica, pero sobre la que posteriormente se puede copiar otra matriz.

```
matrix::matrix(int i, int j) :
        rows_{(i>0)?(i):0},
        cols_{(j>0)?(j):0},
        buffer_{((i==0) && (j==0))?(nullptr):new double[i*j]{}}
{std::cout<< "CONSTRUCTOR POR DEFECTO" << "\n";}
```

- Debe soportar operaciones de copia y movimiento.

```
matrix(const matrix & v);
    //asignacion copia
matrix & operator=(const matrix & v);

    //contructor movimiento
matrix(matrix && m) :
    buffer_{m.buffer_},
    rows_{m.getRows()},
    cols_{m.getCols()}
{std::cout<< "CONSTRUCTOR DE MOVIMIENTO" << "\n";
    m.buffer_ = nullptr;
    m.rows_ = 0;
    m.cols_ = 0; }
    //asignacion movimiento
    matrix & operator=(const matrix && m);
```

- Destructor que debe liberar la memoria que pueda ser propiedad de la matriz.

```
~matrix() { delete []buffer_; }
```

- El operador paréntesis se sobrecargará con dos argumentos para acceder a una posición de la matriz.

```
double operator()(int i, int j) const { return buffer_[i * cols_ + j]; }
```

- Los operadores + y * se sobrecargarán para implementar la suma y el producto de matrices.
  Sobrecarga de operador +

```
 matrix & matrix::operator+(const matrix & m){
        if(rows_ == m.getRows() && cols_ == m.getCols())
                    for(int i = 0 ; i < m.getSize() ; i++){
```

Álvaro Suárez. NIA: 100390789
Adrián Hernández. NIA: 100414379

```
                    this->buffer_[i] = this->buffer_[i] + m.getBuffer()[i];}
            return *this}
```

Sobrecarga de operador *

```
    matrix & matrix::operator*(const  matrix & m){
if(cols_ == m.getRows()) {
    matrix aux{rows_,m.getCols()};
    int z = 0;
    int zIni = 0;
    int k=0;
    for (int i = 0; i < getSize(); i++) {
        zIni = k * aux.cols_;
        z = zIni;
        if((i-1) % aux.cols_ == 0){
           z = zIni;
           i = z;}
        for (int j = 0; j < m.getSize(); j++) {
            if(j==0)k++;
            aux.set(z, aux.get(z) + get(i) * m.get(j));
            z++;
            if((j+1) % aux.cols_ == 0 ){
                i++;
                z = zIni;}}}
    *this = aux;}
return *this;}
```

Evaluación

Se implemento un programa en el cual se generaron tres matrices cuadradas con un valor constante n, con sus valores iniciales a cero, posteriormente se rellenaron con números aleatorios con una distribución normal de media 10.00 y desviación estándar de 2.5, posteriormente se calcula la operación solicitada, midiendo el tiempo de ejecución y luego visualizando el mismo para evaluar la rapidez de este con las diversas modificaciones de parámetros que se exigen, finalmente se calcula e imprime el valor S que es el promedio de los elementos de la matriz resultado. El código implementado es el siguiente:

```
void evaluacion(){
    matrix a{n,n};
    matrix b{n,n};
    matrix c{n,n};
    matrix d{n,n};
    double s = 0;
    std::random_device rd{};
    std::mt19937 gen{rd()};
    std::normal_distribution<> dis(10, 2.5);
    for (int i = 0; i < a.getSize(); ++i) {
        a.set( i , dis(gen) );
        b.set( i , dis(gen) );
        c.set( i , dis(gen) ); }
    auto start = chrono::high_resolution_clock::now();
    d = a + b * c;
    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double> diff = end-start;
    cout << "Tiempo de cálculo de la matriz D = " << diff.count() << endl;
    for (int i = 0; i < d.getSize(); ++i) { s = s + d.get(i); }
    s = s/(d.getSize());
    cout << "\n" << "s = " << s << "\n"; }
```

Álvaro Suárez. NIA: 100390789
Adrián Hernández. NIA: 100414379

Resultados

A continuación, se presentan los distintos resultados obtenidos para los distintos valores de las matrices.

Sin operaciones de movimiento

Para n = 10. Tiempo de cálculo de la matriz D = 1.4169e-05

```
         1,08 msec task-clock                #    0,863 CPUs utilized
            0      context-switches          #    0,000 K/sec
            0      cpu-migrations            #    0,000 K/sec
          135      page-faults               #    0,125 M/sec
    3.003.564      cycles                    #    2,788 GHz
    3.502.119      instructions              #    1,17  insn per cycle
      610.678      branches                  #  566,831 M/sec
       18.727      branch-misses             #    3,07% of all branches

  0,001249079 seconds time elapsed

  0,001322000 seconds user
  0,000000000 seconds sys
```

Para n=100. Tiempo de cálculo de la matriz D = 0.00425548

```
           6,90 msec task-clock                #    0,970 CPUs utilized
              0      context-switches          #    0,000 K/sec
              0      cpu-migrations            #    0,000 K/sec
            252      page-faults               #    0,037 M/sec
     19.258.307      cycles                    #    2,792 GHz
     29.385.917      instructions              #    1,53  insn per cycle
      3.529.262      branches                  #  511,682 M/sec
         73.585      branch-misses             #    2,08% of all branches

     0,007108514 seconds time elapsed

     0,007142000 seconds user
     0,000000000 seconds sys
```

Para n = 1000.

Tiempo de cálculo de la matriz D = 4.00756

```
       4.139,44 msec task-clock                #    1,000 CPUs utilized
             34      context-switches          #    0,008 K/sec
              0      cpu-migrations            #    0,000 K/sec
         13.808      page-faults               #    0,003 M/sec
 11.563.232.202      cycles                    #    2,793 GHz
 19.714.912.115      instructions              #    1,70  insn per cycle
  2.095.590.175      branches                  #  506,250 M/sec
      5.523.027      branch-misses             #    0,26% of all branches

    4,140045731 seconds time elapsed

    4,115739000 seconds user
    0,023998000 seconds sys
```

Con operaciones de movimiento

Para n = 10**.** Tiempo de cálculo de la matriz D = 2.0546e-05

```
         1,12 msec task-clock                #    0,836 CPUs utilized
            0      context-switches          #    0,000 K/sec
            0      cpu-migrations            #    0,000 K/sec
          137      page-faults               #    0,123 M/sec
    3.115.015      cycles                    #    2,787 GHz
    3.501.793      instructions              #    1,12  insn per cycle
      610.572      branches                  #  546,348 M/sec
       18.584      branch-misses             #    3,04% of all branches

  0,001336645 seconds time elapsed

  0,001371000 seconds user
  0,000000000 seconds sys
```

Álvaro Suárez. NIA: 100390789
Adrián Hernández. NIA: 100414379

Para n = 100 Tiempo de cálculo de la matriz D = 0.00457016

```
        7,12 msec task-clock                #    0,971 CPUs utilized
           0      context-switches          #    0,000 K/sec
           0      cpu-migrations            #    0,000 K/sec
         249      page-faults               #    0,035 M/sec
  19.874.239      cycles                    #    2,793 GHz
  29.350.857      instructions              #    1,48  insn per cycle
   3.524.046      branches                  #  495,164 M/sec
      73.052      branch-misses             #    2,07% of all branches

 0,007330963 seconds time elapsed

 0,003681000 seconds user
 0,003681000 seconds sys
```

Para n = 1000. Tiempo de cálculo de la matriz D = 4.25516

```
      4.372,84 msec task-clock                #    1,000 CPUs utilized
             8      context-switches          #    0,002 K/sec
             0      cpu-migrations            #    0,000 K/sec
        13.811      page-faults               #    0,003 M/sec
12.215.411.323      cycles                    #    2,793 GHz
19.712.990.873      instructions              #    1,61  insn per cycle
 2.095.302.075      branches                  #  479,162 M/sec
     5.517.533      branch-misses             #    0,26% of all branches

   4,373133805 seconds time elapsed

   4,349102000 seconds user
   0,023984000 seconds sys
```

Evaluando las distintas ejecuciones se aprecia que con movimiento la ejecución es más rápida. La diferencia se aprecia mejor a medida que se aumenta el valor del tamaño.

En la carpeta de la entrega se encuentran los siguientes archivos:
• Dos carpetas comprimidas con movimiento y sin movimiento
• el archivo de la memoria Laboratorio 1.